The Clojalyzer

Matt Fenwick
Developer, Hudl

Demo

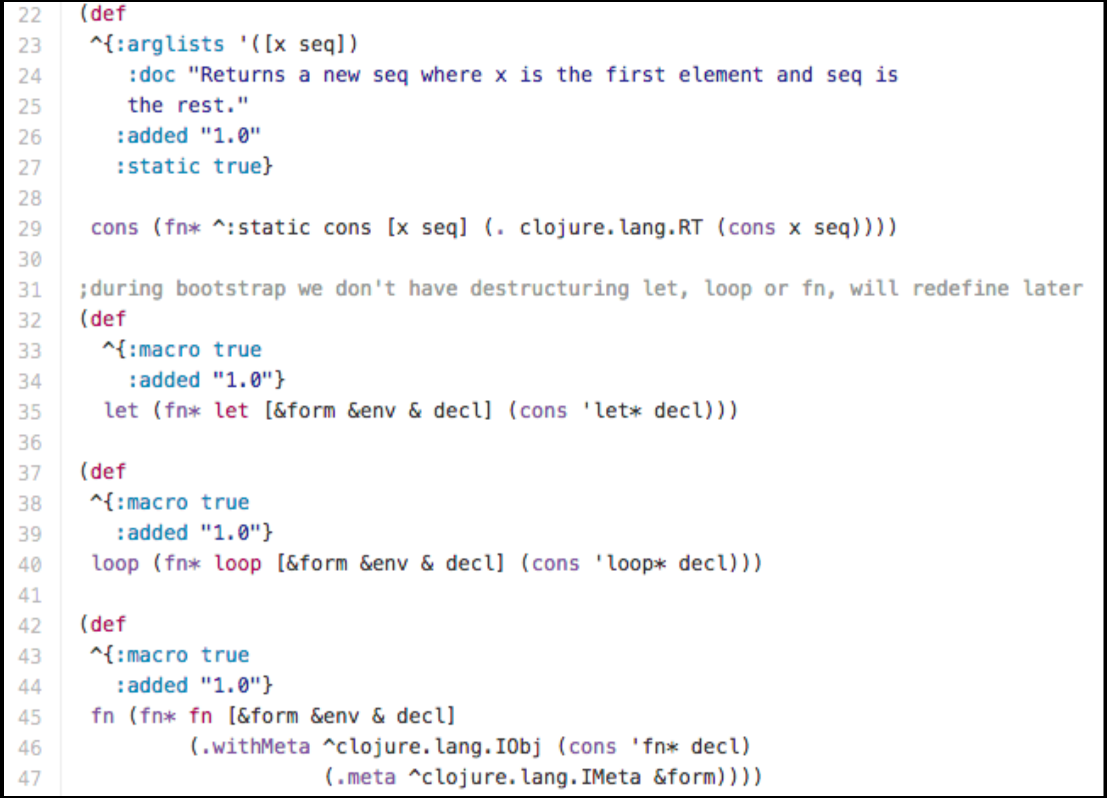
- clojure/Clojure/master
- clojure/Clojurescript/master

free hosting on github.io -- ‘gh-pages’ branch

(https://github.com/mattfenwick/Clojalyzer -> http://mattfenwick.github.io/Clojalyzer/)

Clojure code

https://github.com/clojure/clojure/blob/master/src/clj/clojure/core.clj



```
(def
 ^{:arglists '([x seq])
    :doc "Returns a new seq where x is the first element and seq is
    the rest."
   :added "1.0"
   :static true}

 cons (fn* ^:static cons [x seq] (. clojure.lang.RT (cons x seq))))

;during bootstrap we don't have destructuring let, loop or fn, will redefine later
(def
  ^{:macro true
    :added "1.0"}
  let (fn* let [&form &env & decl] (cons 'let* decl)))

(def
 ^{:macro true
   :added "1.0"}
 loop (fn* loop [&form &env & decl] (cons 'loop* decl)))

(def
 ^{:macro true
   :added "1.0"}
 fn (fn* fn [&form &env & decl]
         (.withMeta ^clojure.lang.IObj (cons 'fn* decl)
                    (.meta ^clojure.lang.IMeta &form))))
```

Clojalyzer: architecture

Heavy lifting

- Clojarse-js
- UnParse-js

Check out https://www.npmjs.com/~mattfenwick to find this package on NPM!

Frontend

- browserify
- jQuery
- Github API
- glue

Clojalyzer: workflow

1. input: string
2. build a concrete syntax tree (CST)
   - must handle syntax errors gracefully
3. map CST to abstract syntax tree (AST)
   - get rid of unnecessary details in CST
4. run queries on AST
5. graph, graph queries

```
(do
  (let [y 3] y)
  (let [z 4]
    (+ w 2)))
```

Caveats

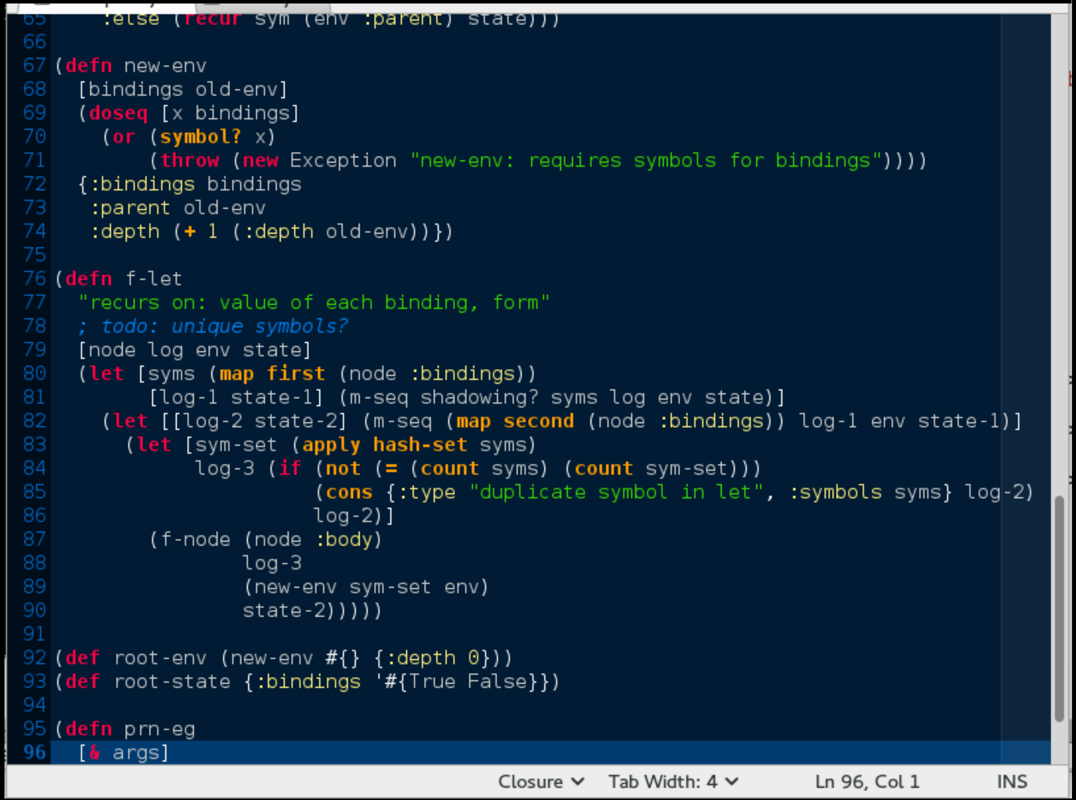
Parsing clojure 100% accurately is really hard!

It’s not really possible to do static analysis!

A lot of syntax is implementation-defined, and the implementations disagree .

gedit-clojure: syntax highlighting for gedit

https://github.com/mattfenwick/gedit-clojure



```
    :else (recur sym (env :parent) state)))

(defn new-env
  [bindings old-env]
  (doseq [x bindings]
    (or (symbol? x)
        (throw (new Exception "new-env: requires symbols for bindings"))))
  {:bindings bindings
   :parent old-env
   :depth (+ 1 (:depth old-env))})

(defn f-let
  "recurs on: value of each binding, form"
  ; todo: unique symbols?
  [node log env state]
  (let [syms (map first (node :bindings))
        [log-1 state-1] (m-seq shadowing? syms log env state)]
    (let [[log-2 state-2] (m-seq (map second (node :bindings)) log-1 env state-1)]
      (let [sym-set (apply hash-set syms)
            log-3 (if (not (= (count syms) (count sym-set)))
                      (cons {:type "duplicate symbol in let", :symbols syms} log-2)
                      log-2)]
        (f-node (node :body)
                log-3
                (new-env sym-set env)
                state-2)))))

(def root-env (new-env #{} {:depth 0}))
(def root-state {:bindings '#{True False}})

(defn prn-eg
  [& args]
```

Closure Tab Width: 4 Ln 96, Col 1 INS

Questions?